lab-5-rachit

May 7, 2025

[ ]:
```
import cv2
import numpy as np
from PIL import Image
import io
import os

from google.colab import files
uploaded = files.upload()

filename = list(uploaded.keys())[0]

image = cv2.imread(filename)


image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

original_size = os.path.getsize(filename) / 1024
print(f"Original Image Size: {original_size:.2f} KB")


jpeg_filename = "compressed_image.jpg"
cv2.imwrite(jpeg_filename, image, [cv2.IMWRITE_JPEG_QUALITY, 30])


jpeg_size = os.path.getsize(jpeg_filename) / 1024
print(f"JPEG Compressed Size: {jpeg_size:.2f} KB")


png_filename = "compressed_image.png"
cv2.imwrite(png_filename, image, [cv2.IMWRITE_PNG_COMPRESSION, 9])


png_size = os.path.getsize(png_filename) / 1024
print(f"PNG Compressed Size: {png_size:.2f} KB")


import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots(1, 3, figsize=(15,5))


ax[0].imshow(image_rgb)
ax[0].set_title("Original Image")
ax[0].axis("off")


jpeg_img = Image.open(jpeg_filename)
ax[1].imshow(jpeg_img)
ax[1].set_title("JPEG Compressed Image")
ax[1].axis("off")


png_img = Image.open(png_filename)
ax[2].imshow(png_img)
ax[2].set_title("PNG Compressed Image")
ax[2].axis("off")

plt.show()
```

```
<IPython.core.display.HTML object>
```

```
Saving car.jpeg to car.jpeg
Original Image Size: 9.43 KB
JPEG Compressed Size: 5.09 KB
PNG Compressed Size: 78.47 KB
```

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import classification_report, roc_auc_score, roc_curve,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape to match CNN input (28x28 images with 1 channel)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434              2s
0us/step
```

[ ]:
```
# Define CNN model
model = keras.Sequential([
    layers.Conv2D(32, (3,3), strides=2, activation='relu',␣
 ↪input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), strides=2, activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[ ]:
```
history = model.fit(x_train, y_train, epochs=15, batch_size=128,␣
 ↪validation_split=0.2)
```

```
Epoch 1/15
375/375            7s 7ms/step -
```

```
accuracy: 0.6835 - loss: 1.1365 - val_accuracy: 0.9277 - val_loss: 0.2477
Epoch 2/15
375/375             6s 4ms/step -
accuracy: 0.9370 - loss: 0.2075 - val_accuracy: 0.9494 - val_loss: 0.1677
Epoch 3/15
375/375             3s 4ms/step -
accuracy: 0.9525 - loss: 0.1561 - val_accuracy: 0.9643 - val_loss: 0.1188
Epoch 4/15
375/375             2s 4ms/step -
accuracy: 0.9625 - loss: 0.1232 - val_accuracy: 0.9632 - val_loss: 0.1192
Epoch 5/15
375/375             1s 4ms/step -
accuracy: 0.9653 - loss: 0.1120 - val_accuracy: 0.9677 - val_loss: 0.1055
Epoch 6/15
375/375             1s 4ms/step -
accuracy: 0.9712 - loss: 0.0940 - val_accuracy: 0.9694 - val_loss: 0.1045
Epoch 7/15
375/375             1s 4ms/step -
accuracy: 0.9727 - loss: 0.0863 - val_accuracy: 0.9738 - val_loss: 0.0897
Epoch 8/15
375/375             1s 3ms/step -
accuracy: 0.9766 - loss: 0.0766 - val_accuracy: 0.9718 - val_loss: 0.0955
Epoch 9/15
375/375             3s 5ms/step -
accuracy: 0.9778 - loss: 0.0694 - val_accuracy: 0.9705 - val_loss: 0.0965
Epoch 10/15
375/375             2s 4ms/step -
accuracy: 0.9792 - loss: 0.0644 - val_accuracy: 0.9729 - val_loss: 0.0886
Epoch 11/15
375/375             3s 4ms/step -
accuracy: 0.9820 - loss: 0.0599 - val_accuracy: 0.9773 - val_loss: 0.0788
Epoch 12/15
375/375             2s 5ms/step -
accuracy: 0.9824 - loss: 0.0544 - val_accuracy: 0.9758 - val_loss: 0.0813
Epoch 13/15
375/375             1s 4ms/step -
accuracy: 0.9839 - loss: 0.0520 - val_accuracy: 0.9762 - val_loss: 0.0786
Epoch 14/15
375/375             1s 3ms/step -
accuracy: 0.9834 - loss: 0.0508 - val_accuracy: 0.9780 - val_loss: 0.0748
Epoch 15/15
375/375             1s 3ms/step -
accuracy: 0.9869 - loss: 0.0410 - val_accuracy: 0.9778 - val_loss: 0.0783
```

[ ]:
```python
# Predictions
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
```

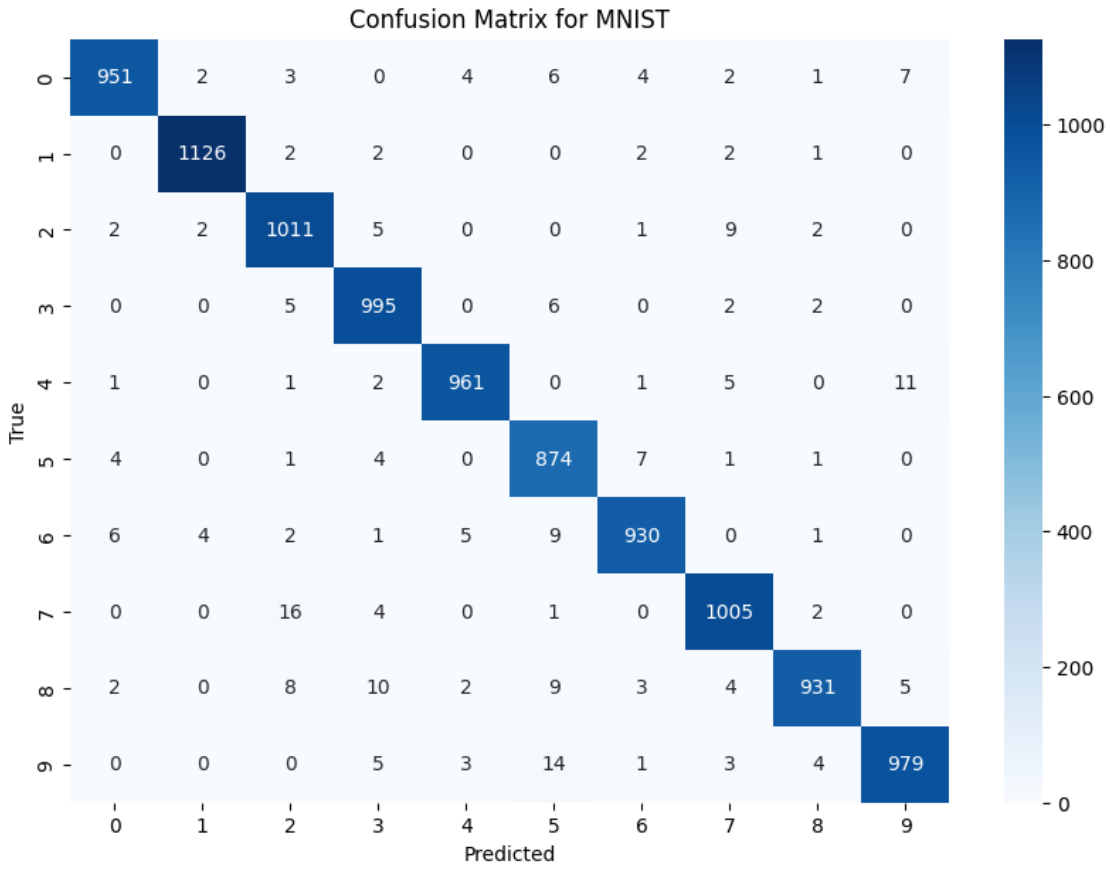
```
y_true = np.argmax(y_test, axis=1)

# Model Evaluation
accuracy = np.mean(y_pred_classes == y_true)
precision = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['precision']
recall = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['recall']
f1_score = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['f1-score']

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for MNIST')
plt.show()

# ROC Curve & AUC
auc_score = roc_auc_score(y_test, y_pred, multi_class='ovr')
print(f"AUC Score: {auc_score:.4f}")
```

```
313/313             1s 3ms/step
```




```
AUC Score: 0.9996
```

[ ]:
```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Normalize pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encode labels
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071              13s
0us/step
```

[ ]:
```
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
```

```
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 classes
])


model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[ ]:
```
history = model.fit(x_train, y_train, epochs=15, batch_size=128,␣
↪validation_split=0.2)
```

```
Epoch 1/15
313/313             8s 14ms/step -
accuracy: 0.2678 - loss: 1.9606 - val_accuracy: 0.4596 - val_loss: 1.4896
Epoch 2/15
313/313             6s 6ms/step -
accuracy: 0.4892 - loss: 1.3996 - val_accuracy: 0.5460 - val_loss: 1.2645
Epoch 3/15
313/313             2s 6ms/step -
accuracy: 0.5656 - loss: 1.2113 - val_accuracy: 0.5886 - val_loss: 1.1689
Epoch 4/15
313/313             3s 6ms/step -
accuracy: 0.6074 - loss: 1.1219 - val_accuracy: 0.5945 - val_loss: 1.1452
Epoch 5/15
313/313             3s 7ms/step -
accuracy: 0.6399 - loss: 1.0287 - val_accuracy: 0.6380 - val_loss: 1.0438
Epoch 6/15
313/313             2s 6ms/step -
accuracy: 0.6637 - loss: 0.9611 - val_accuracy: 0.6526 - val_loss: 1.0037
Epoch 7/15
313/313             2s 6ms/step -
accuracy: 0.6907 - loss: 0.8921 - val_accuracy: 0.6774 - val_loss: 0.9399
Epoch 8/15
313/313             2s 6ms/step -
accuracy: 0.7097 - loss: 0.8351 - val_accuracy: 0.6752 - val_loss: 0.9519
Epoch 9/15
```

```
313/313             2s 6ms/step -
accuracy: 0.7235 - loss: 0.7937 - val_accuracy: 0.6858 - val_loss: 0.9179
Epoch 10/15
313/313             2s 6ms/step -
accuracy: 0.7479 - loss: 0.7377 - val_accuracy: 0.6724 - val_loss: 0.9573
Epoch 11/15
313/313             3s 7ms/step -
accuracy: 0.7533 - loss: 0.7100 - val_accuracy: 0.6935 - val_loss: 0.9000
Epoch 12/15
313/313             2s 5ms/step -
accuracy: 0.7701 - loss: 0.6640 - val_accuracy: 0.7068 - val_loss: 0.8738
Epoch 13/15
313/313             3s 6ms/step -
accuracy: 0.7838 - loss: 0.6274 - val_accuracy: 0.7128 - val_loss: 0.8671
Epoch 14/15
313/313             2s 6ms/step -
accuracy: 0.7948 - loss: 0.6005 - val_accuracy: 0.7085 - val_loss: 0.8717
Epoch 15/15
313/313             2s 5ms/step -
accuracy: 0.8045 - loss: 0.5589 - val_accuracy: 0.7086 - val_loss: 0.9010
```

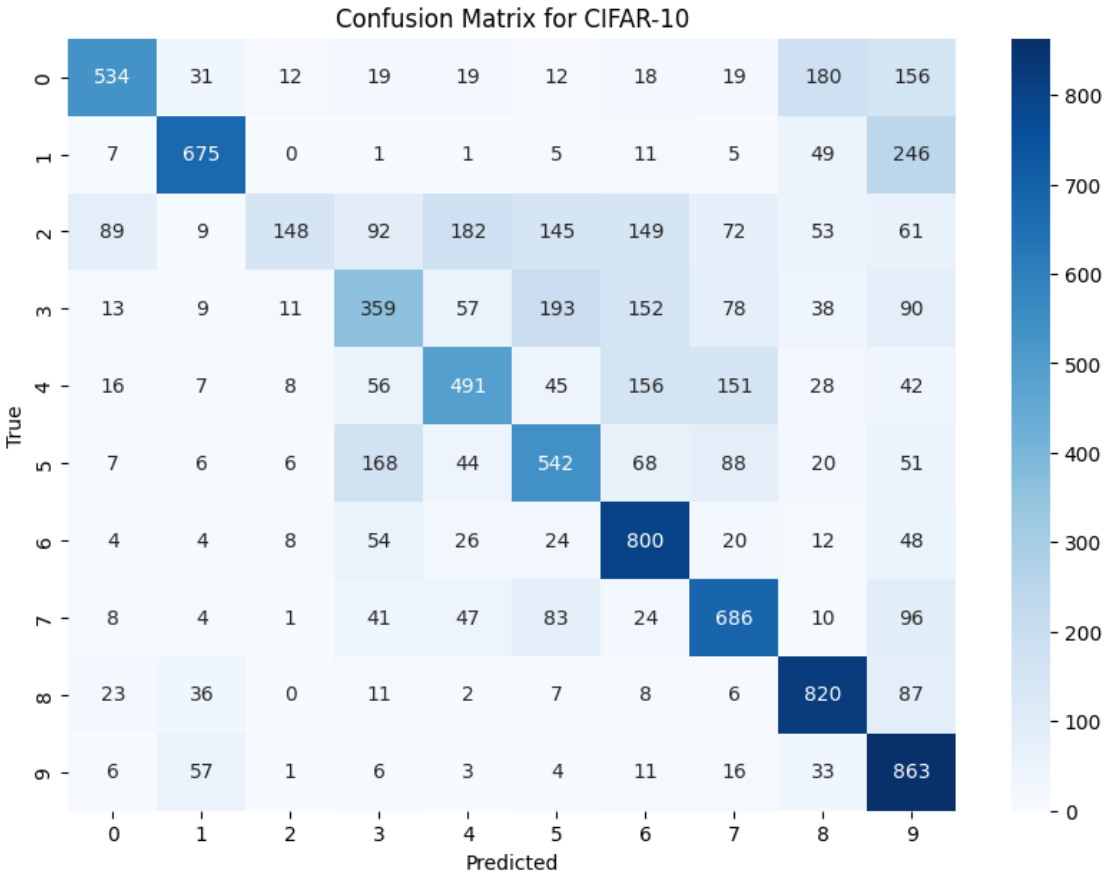
[ ]:
```python
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)


accuracy = np.mean(y_pred_classes == y_true)
precision = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['precision']
recall = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['recall']
f1_score = classification_report(y_true, y_pred_classes,␣
 ↪output_dict=True)['weighted avg']['f1-score']


conf_matrix = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for CIFAR-10')
plt.show()


auc_score = roc_auc_score(y_test, y_pred, multi_class='ovr')
print(f"AUC Score: {auc_score:.4f}")
```

```
313/313                3s 11ms/step
```




```
AUC Score: 0.9290
```